

Sensor Management for Tracking in Sensor Networks

Jason A. Fuemmeler, *Member, IEEE*, George K. Atia, *Member, IEEE*, and
Venugopal V. Veeravalli, *Fellow, IEEE*

Abstract

We study the problem of tracking an object moving through a network of wireless sensors. In order to conserve energy, the sensors may be put into a sleep mode with a timer that determines their sleep duration. It is assumed that an asleep sensor cannot be communicated with or woken up, and hence the sleep duration needs to be determined at the time the sensor goes to sleep based on all the information available to the sensor. Having sleeping sensors in the network could result in degraded tracking performance, therefore, there is a tradeoff between energy usage and tracking performance. We design sleeping policies that attempt to optimize this tradeoff and characterize their performance. As an extension to our previous work in this area [1], we consider generalized models for object movement, object sensing, and tracking cost. For discrete state spaces and continuous Gaussian observations, we derive a lower bound on the optimal energy-tracking tradeoff. It is shown that in the low tracking error regime, the generated policies approach the derived lower bound.

I. INTRODUCTION

Large sensor networks collecting data in dynamic environments are typically composed of a distributed collection of cheap nodes with limited energy and processing capabilities. Hence, it is imperative to efficiently manage the sensors' resources to prolong the lifetime of such networks without sacrificing performance. Our focus in this paper is on sensor resource management for tracking and surveillance applications.

This work was funded in part by a grant from the Motorola corporation, a U.S. Army Research Office MURI grant W911NF-06-1-0094 through a subcontract from Brown University at the University of Illinois, a NSF Graduate Research Fellowship, and by a Vodafone Fellowship.

This work was done at the Coordinated Science Laboratory (CSL), University of Illinois at Urbana-Champaign, Urbana IL 61801, Emails: {fuemmele, atia1, vv} @illinois.edu

Previous work on sensor resource management considered the design of sensor sleeping protocols for sensor sleeping via wakeup mechanisms [2]–[7] or by modifying power-save functions in MAC protocols for wireless ad hoc networks [8]–[10]. In the context of target classification, Castanon [11] developed an approximate dynamic programming approach for dynamic scheduling of multi-mode sensors subject to sensors resource constraints. In [12], [13] we studied a single object tracking problem where the sensors can be turned on or off at consecutive time steps to conserve energy (sensor scheduling). A controller selects the subset of sensors to activate at each time step. Also in [1], we studied a tracking problem where each sensor could enter a sleep mode with a sleep timer (sensor sleeping). While in sleep mode, the sensor could not assist in tracking the object by making observations. In contrast to [13], in [1] we assumed that sleeping sensors could not be woken up externally but instead had to set internal timers to determine the next time to come awake, wherefore, the control actions correspond to the sleep durations of awake sensors. In turn, this did not only entail a different control space, but also led to a significantly different policy design problem since a decision to put a sensor to sleep implies that this sensor cannot be scheduled at future time steps until it comes awake. The consequences of the current action on the tracking performance could be more dramatic rendering future planning more crucial. This led to a design problem that sought to optimize a tradeoff between energy efficiency and tracking performance. While optimal solutions to this problem could not be found, suboptimal solutions were devised that were demonstrated to be near optimal. To aid analysis, we assumed particularly simple models for object movement, object sensing, and tracking cost. In particular, we assumed that the network could be divided into cells, each of which contained a single sensor. The object moved among the cells and could only be observed by the sensor in the currently occupied cell. Tracking performance was a binary quantity; either the object was observed in a particular time slot or it was not observed depending on whether the right sensor was awake.

In this paper, we continue to examine the fundamental theory of sleeping in sensor networks for tracking but we extend our analysis to more generalized models for object movement, object sensing, and tracking cost. We allow the number of possible object locations to be different from the number of sensors. The number of possible object locations can even be infinite to model the movement of an object on a continuum. Moreover, the object sensing model allows for an arbitrary distribution for the observations given the current object location, and the tracking cost is modeled via an arbitrary distance measure between the actual and estimated object location.

Not surprisingly, this generalization results in a problem that is much more difficult to analyze. Our approach is to build on the policies designed in [1]. The design of those policies relied on the separation

of the problem into a set of simpler subproblems. In [1], we have shown that under an observable-after-control assumption, the design problem lends itself to a natural decomposition into simpler per-sensor subproblems due to the simplified nature of the tracking cost structure. Unfortunately, this does not extend to the generalized cases we consider herein. However, based on the intuition gained from the structure of the solution in the simplified case, in this work we artificially separate our problem into a set of simpler per-sensor subproblems. The parameters of these subproblems are not known *a priori* due to the difficulties in analysis. However, we use Monte Carlo simulation and learning algorithms to compute these parameters. We characterize the performance of the resulting sleeping policies through simulation. For the special case of a discrete state space with continuous Gaussian observations, we derive a lower bound on the optimal energy-tracking tradeoff which is shown to be loose at the high tracking error regime, but is reasonably tight for the low tracking error region.

The remainder of this paper is organized as follows. In Section II, we describe the tracking problem in mathematical terms and define the optimization problem. In Section III we derive our suboptimal solutions and the aforementioned lower bound. In Section IV, we provide numerical results that illustrate the efficacy of the proposed sleeping policies. We summarize and conclude in Section V.

II. PROBLEM FORMULATION

A. POMDP Formulation

Consider a network with n sensors. Each sensor can be in one of two states: awake or asleep. A sensor in the awake state consumes more energy than one in the asleep state. However, object sensing can be performed only in the awake state. We denote the set of possible object locations as \mathcal{B} such that $|\mathcal{B}| = m + 1$ where the $(m + 1)$ -th state represents an absorbing terminal state that occurs when the object leaves the network. We also refer to this terminal state as \mathcal{T} . If \mathcal{B} is not a finite set then m is ∞ . We define a *kernel* P such that $P(x, \mathcal{Y})$ is the probability that the next object location is in the set $\mathcal{Y} \subset \mathcal{B}$ given that the current object location is x . We can predict t time steps into the future by defining $P^1 = P$ and P^t inductively as

$$P^t(x, \mathcal{Y}) = \int_{\mathcal{B}} P^{t-1}(x, dz) P(z, \mathcal{Y}) \quad (1)$$

Suppose p is a probability measure on \mathcal{B} such that $p(\mathcal{X})$ for $\mathcal{X} \in \mathcal{B}$ is the probability that the state is in \mathcal{X} at the current time step. Then the probability that the state will be in \mathcal{Y} after t time steps in the future is given by

$$(pP^t)(\mathcal{Y}) \equiv \int_{\mathcal{B}} p(dx) P^t(x, \mathcal{Y}) \quad (2)$$

This defines the measure pP^t which depends on both the prior p and the transition Kernel P . Let b_k denote the state for the object at time k . Also, let δ_x denote a probability measure such that $\delta_x(\mathcal{A}) = 1$ if $x \in \mathcal{A}$, and $\delta_x(\mathcal{A}) = 0$ otherwise. Conditioned on the object state b_k , the future state b_{k+1} has a distribution $\delta_{b_k}P$. This defines the evolution of the object location. For a discrete state space this is simply the probability mass function defined by the b_k -th row of a transition matrix P . We assume that it is always possible to determine if the object has left the network, i.e., if $b_k = m + 1$. To this end, we define a virtual sensor $n + 1$ that detects without error whether the object has left the network. In other words, sensor $n + 1$ is always awake but consumes no energy.

To provide a means for centralized control, we assume the presence of an extra node called the central controller. The central controller keeps track of the state of the network and assigns sleep times to sensors that are awake. In particular, each sensor that wakes up remains awake for one time unit during which the following actions are taken: (i) the sensor sends its observation of the object to the central unit, and (ii) the sensor receives a new sleep time (which may equal zero) from the central controller. The sleep time input is used to initialize a timer at the sensor that is decremented by one time unit each time step. When this timer expires, the sensor wakes up. Since we assume that wakeup signals are impractical, this timer expiration is the only mechanism for waking a sensor.

Let $r_{k,\ell}$ denote the value of the sleep timer of sensor ℓ at time k . We call the $(n + 1)$ -vector \mathbf{r}_k the residual sleep times of the sensors at time k . Also, let $u_{k,\ell}$ denote the sleep time input supplied to sensor ℓ at time k . We add the constraints $r_{k,n+1} = 0$ and $u_{k,n+1} = 0$ due to the nature of the virtual sensor $n + 1$. We can describe the evolution of the residual sleep times as

$$r_{k+1,\ell} = (r_{k,\ell} - 1)\mathbb{I}\{r_{k,\ell} > 0\} + u_{k,\ell}\mathbb{I}\{r_{k,\ell} = 0\} \quad (3)$$

for all k and $\ell \in \{1, \dots, n + 1\}$. The first term on the right hand side of this equation expresses that if the sensor is currently asleep (the sleep timer for the sensor is not zero), the sleep timer is decremented by 1. The second term expresses that if the sensor is currently awake (the sleep timer is zero), the sleep timer is reset to the current sleep time input for that sensor.

Based on the probabilistic evolution of the object location and (3), we see that we have a discrete-time dynamical model that describes our system with a well-defined state evolution. The *state* of the system at time k is described by $x_k = (b_k, \mathbf{r}_k)$. Unfortunately, not all of x_k is known to the central unit at time k since b_k is known only if the object location is being tracked precisely. Thus we have a dynamical system with incomplete (or partially observed) state information.

We write the observations for our problem as

$$z_k = (\mathbf{s}_k, \mathbf{r}_k) \quad (4)$$

where \mathbf{s}_k is an $(n+1)$ -vector of observations. These observations are drawn from a probability measure σ_{x_k} that depends on x_k . However, we add two restrictions. The first is that if a sensor is not awake at time k , its observation is an erasure. Mathematically, we say that $r_{k,\ell} > 0$ implies $s_{k,\ell} = \mathcal{E}$. The second restriction is that $s_{k,n+1}$ is a binary observation that indicates whether the object has left the network.

The total information available to the control unit at time k is given by

$$I_k = (z_0, \dots, z_k, \mathbf{u}_0, \dots, \mathbf{u}_{k-1}) \quad (5)$$

with $I_0 = z_0$ denoting the initial (known) state of the system. The control input for sensor ℓ at time k is allowed to be a function of the information state I_k , i.e.,

$$\mathbf{u}_k = \mu_k(I_k) \quad (6)$$

The vector-valued function μ_k is the sleeping policy at time k which defines a mapping from the information state I_k to the set of admissible actions \mathbf{u}_k .

We now identify the costs present in our tracking problem. The first is an *energy cost* of $c > 0$ for each sensor that is awake. The energy cost can be written mathematically as

$$\sum_{\ell=1}^n c \mathbb{1}\{r_{k,\ell} = 0\} \quad (7)$$

The second cost is a *tracking cost*. To define the tracking cost, we first define the estimated object location at time k to be \hat{b}_k . We can think of \hat{b}_k as an additional control input that is a function of I_k , i.e.,

$$\hat{b}_k = \beta_k(I_k) \quad (8)$$

Since \hat{b}_k does not affect the state evolution, we do not need past values of this control input in I_k . The tracking cost is a distance measure that is a function of the actual and estimated object locations and is written as

$$d(b_k, \hat{b}_k) \quad (9)$$

We assume that d is a bounded function on $\mathcal{B} \times \mathcal{B}$. Two examples of distance measures we might employ are the Hamming cost (if the space \mathcal{B} is finite), i.e.,

$$d(b_k, \hat{b}_k) = \mathbb{1}\{\hat{b}_k \neq b_k\} \quad (10)$$

and the squared Euclidean distance (if the space \mathcal{B} is a subset of an appropriate vector space), i.e.,

$$d(b_k, \hat{b}_k) = \|\hat{b}_k - b_k\|_2^2 \quad (11)$$

The parameter c is used to trade off energy consumption and tracking errors.

Recall that the input \hat{b}_k does not affect the state evolution; it only affects the cost. Therefore, we can compute the optimal choice of \hat{b}_k , given by $\beta_k^*(I_k)$, using an optimization minimizing the tracking error over a single time step. We can thus write

$$\beta_k^*(I_k) = \arg \min_{\hat{b}} \mathbf{E} \left[d(b_k, \hat{b}_k) \middle| I_k \right] \quad (12)$$

Remembering that once the terminal state is reached no further cost is incurred, we can write the total cost for time step k as

$$g(b_k, I_k) = \mathbb{1}\{b_k \neq \mathcal{T}\} \left(d(b_k, \beta_k^*(I_k)) + \sum_{\ell=1}^n c \mathbb{1}\{r_{k,\ell} = 0\} \right) \quad (13)$$

The infinite horizon cost for the system is given by

$$J(I_0, \mu_0, \mu_1, \dots) = \mathbf{E} \left[\sum_{k=1}^{\infty} g(b_k, I_k) \middle| I_0 \right] \quad (14)$$

Since g is bounded (since the function d is bounded) and the expected time till the object leaves the network is finite, the cost function J is well defined. The goal is to compute the solution to

$$J^*(I_0) = \min_{\mu_0, \mu_1, \dots} J(I_0, \mu_0, \mu_1, \dots) \quad (15)$$

The solution to this optimization problem for each value of c yields an optimal sleeping policy. The optimization problem falls under the framework of a partially observable Markov decision process (POMDP) [14]–[17].

B. Dealing With Partial Observability

Partial observability presents a problem since the information for decision-making at time k given in (5) is unbounded in memory. To remedy this, we seek a sufficient statistic for optimization that is bounded in memory. The observation s_k depends only on x_k , which in turn depends only on x_{k-1} , u_{k-1} , and some random disturbance w_{k-1} . It is a standard argument (e.g., see [18]) that for such an observation model, a sufficient statistic is given by the probability distribution of the state x_k given I_k . Such a sufficient statistic is referred to as a *belief state* in the POMDP literature (e.g., see [14], [15]).

Since the residual sleep times portion of our state is observable, the sufficient statistic can be written as $v_k = (p_k, \mathbf{r}_k)$, where p_k is a probability measure on \mathcal{B} . Mathematically, we have

$$p_k(\mathcal{X}) = \mathbf{P}(b_k \in \mathcal{X} | I_k) \quad (16)$$

The task of recursively computing p_k for each k is a problem in nonlinear filtering (e.g., see [19]). In other words, p_{k+1} can be computed using standard Bayesian techniques as the posterior measure resulting from prior measure pP and observations \mathbf{s}_{k+1} .

The function β_k^* that determines \hat{b}_k can now be written in terms of p_k and \mathbf{r}_k instead of I_k . We can rewrite it as

$$\beta_k^*(p_k, \mathbf{r}_k) = \arg \min_{\hat{b}} \mathbf{E} \left[d(b_k, \hat{b}) | b_k \sim p_k \right] \quad (17)$$

$$= \arg \min_{\hat{b}} \int_{\mathcal{B}} d(b_k, \hat{b}) p_k(db) \quad (18)$$

Note that due to the stationarity of the state evolution, β_k^* has the same form for every k and is independent of \mathbf{r}_k . Thus, we can drop the subscript and refer to β_k^* as β^* , a function of p_k alone.

Now we write our dynamic programming problem in terms of the sufficient statistic. We first rewrite the cost at time step k . Since only expected values of the cost function g appear in (14), we can take our cost function to be the expected value of g (defined in (13)) conditioned on b_k being distributed according to p_k . With a slight abuse of notation, we call this redefined cost g . The cost can then be written as

$$g(p_k, \mathbf{r}_k) = \int_{\mathcal{B}} \mathbb{1}\{b \neq \mathcal{T}\} \left(d(b, \beta^*(p_k)) + \sum_{\ell=1}^n c \mathbb{1}\{r_{k,\ell} = 0\} \right) p_k(db) \quad (19)$$

$$= \int_{\mathcal{B}-\mathcal{T}} \left(d(b, \beta^*(p_k)) + \sum_{\ell=1}^n c \mathbb{1}\{r_{k,\ell} = 0\} \right) p_k(db) \quad (20)$$

The selection of sleep times, originally presented in (6), can now be rewritten as

$$\mathbf{u}_k = \mu_k(p_k, \mathbf{r}_k) \quad (21)$$

The total cost defined in (14) becomes

$$J(p_0, \mathbf{r}_0, \mu_0, \mu_1, \dots) = \mathbf{E} \left[\sum_{k=1}^{\infty} g(p_k, \mathbf{r}_k) \middle| v_0 \right] \quad (22)$$

and the optimal cost defined in (15) becomes

$$J^*(p_0, \mathbf{r}_0) = \min_{\mu_0, \mu_1, \dots} J(p_0, \mathbf{r}_0, \mu_0, \mu_1, \dots) \quad (23)$$

III. SUBOPTIMAL SOLUTIONS

Similar to the problem in [1], an optimal policy could be found by solving the Bellman equation

$$J(p, \mathbf{r}) = \min_{\mu} \mathbf{E} [g(p_1, \mathbf{r}_1) + J(p_1, \mathbf{r}_1) | p_0 = p, \mathbf{r}_0 = \mathbf{r}, \mathbf{u}_0 = \mu(p_0, \mathbf{r}_0)] \quad (24)$$

However, since an optimal solution could not be found for the simpler problem considered in [1], we immediately turn our attention to finding suboptimal solutions to our problem.

Note that in [1], simpler sensing models and cost structures were employed. Under a simplifying observable-after-control assumption, the simplicity of the sensing models allowed for the decoupling of the contributions of the individual sensors. The simplicity of the cost structures allowed the cost to be written as a sum of per-sensor costs. The result was a problem that could be written as a number of simpler subproblems. The present case is more complicated. In general, the cooperation among the sensors may be difficult to analyze and understand. Furthermore, the tracking cost may not be easily written as a sum across the sensors.

Based on the intuition gained from [1], our approach to generating suboptimal solutions is to artificially write the problem as a set of subproblems that can be solved using the techniques of [1]. The tracking cost expressions (which are a function of the sleeping actions of the sensors) in these subproblems will be left as unknowns. To determine appropriate values for these tracking costs, we either perform Monte Carlo simulations before tracking begins or use data gathered during tracking. The intuition is that if the resultant tracking cost expressions capture the “typical” behavior of the actual tracking cost, then our sleeping policies should perform well.

A. General approach

The complexity of the sleeping problem stems from:

- 1) The complicated evolution of the belief state p_k (non-linear filtering).
- 2) The complexity of the model including the dimensionality of the state space, the control space and the observation space.

To address the aforementioned difficulties, our approach has two main ingredients. First, we make assumptions about the observations that will be available to the controller at future time steps. To generate sleeping policies, we assume that the system is either perfectly observable or totally unobservable after control. Hence, we define approximate recursions with special structure as surrogates for the optimal value function. Second, we devise different methodologies to evaluate suitable tracking costs in Sections III-B and III-C whereby we capture the effect of each sensor on the overall tracking cost. Writing the

combined tracking cost as the sum of independent contributions of different sensors (with respect to some baseline) allows us to write the Bellman equation as the sum of per-sensor recursions. Instead of solving the Bellman equation in (24), we alternatively solve n simpler Bellman equations to find per-sensor policies and cost functions. The overall policy is then the per-sensor policies applied in parallel.

We denote by $J^{(\ell)}$ the cost function of the ℓ -th sensor approximate subproblem. We define $T^\Delta(b, \ell)$ to be the increase in tracking cost due to not waking up sensor ℓ at time k given that $b_{k-1} = b$. This is meant to capture the contribution of the ℓ -th sensor to the total tracking cost. Next we define our approximations.

1) Q_{MDP} : First introduced in the artificial intelligence literature [20], [21], the Q_{MDP} solution for POMDPs assumes that the system will be perfectly observable after control, i.e., the partially observable state becomes fully observable after taking a control action. In other words, under a Q_{MDP} assumption the belief state simply evolves as

$$p_{k+1} = \delta_{b_{k+1}} \quad (25)$$

Noting that the future cost is not only affected by the current control action through belief evolution, but also by the fact that no future decisions can be made for a sleeping sensor until it wakes up, the observable-after-control policy is by no means a myopic policy. Note that (25) does not imply zero tracking errors; it is merely an assumption simplifying the state evolution in order to generate a sleeping policy. Now we can readily define a Q_{MDP} per-sensor Bellman equation analogous to the one in [1] as

$$J^{(\ell)}(p) = \min_u \left(\sum_{j=0}^{u-1} \int_{\mathcal{B}-\mathcal{T}} T^\Delta(b, \ell) (pP^j)(db) + \int_{\mathcal{B}-\mathcal{T}} \left(c + J^{(\ell)}(\delta_b) \right) (pP^{u+1})(db) \right) \quad (26)$$

To clarify, the first summation in the R.H.S. of (26) corresponds to the expected tracking cost incurred by the sleep duration u of sensor ℓ . The second term consists of: (i) the energy cost incurred as the sensor comes awake after its sleep timer expires (after $u + 1$ time slots); and (ii) the cost to go under an observable-after-control assumption (hence the belief state is δ_b).

We cannot find an analytical solution for (26). However, note that if we can solve (26) for $p = \delta_b$ for all b , then it is straightforward to find the solution for all values of p . Thus, given a function T^Δ , (26) can be solved through standard policy iteration [18], but only if \mathcal{B} is finite.

2) FCR : Similarly, we define a First Cost Reduction (FCR) Bellman equation analogous to the one in [1] as

$$J^{(\ell)}(p) = \min_u \left(\sum_{j=0}^{u-1} \int_{\mathcal{B}-\mathcal{T}} T^\Delta(b, \ell) (pP^j)(db) + c \int_{\mathcal{B}-\mathcal{T}} (pP^{u+1})(db) + J^{(\ell)}(pP^{u+1}) \right) \quad (27)$$

In this case, it is assumed that we will have no future observations. In other words, we define the belief evolution as $p_{k+1} = p_k P$. Again, it is worth mentioning that this does not mean that it would be impossible to track the object; we are simply making a simplifying assumption about the future state evolution in order to generate a sleeping policy. Given a function T^Δ , it is easy to verify that the solution to (27) is

$$J^{(\ell)}(p) = \sum_{j=0}^{\infty} \min \left\{ \int_{\mathcal{B}-\mathcal{T}} T^\Delta(b, \ell) (pP^j)(db), c \int_{\mathcal{B}-\mathcal{T}} (pP^{j+1})(db) \right\} \quad (28)$$

and the associated policy is to choose the first value of u such that

$$c \int_{\mathcal{B}-\mathcal{T}} (pP^{u+1})(db) \geq \int_{\mathcal{B}-\mathcal{T}} T^\Delta(b, \ell) (pP^u)(db), \quad (29)$$

In other words, the policy is to come awake at the first time the expected tracking cost exceeds the expected energy cost where the tracking cost is defined based on T^Δ (to be determined) hence the name First Cost Reduction.

The solutions to the per-sensor Bellman equations in (26) and (27) define the Q_{MDP} and FCR policies for each sensor, respectively. Note that, unlike [1], [12], [13], the solution to the Q_{MDP} recursion does not necessarily provide a lower bound on the optimal value function since the employed tracking cost is not a lower bound on the actual tracking cost. In Sec III-D we derive a lower bound on the optimal energy-tracking tradeoff for discrete state spaces with Gaussian Observations. The remaining task is to identify appropriate values of $T^\Delta(b, \ell)$ for all $b \neq \mathcal{T}$ and for all ℓ . This is the subject of the next two sections.

B. Nonlearning approach

For now, suppose that \mathcal{B} is a finite space. Suppose $b_{k-1} = b$. To generate $T^\Delta(b, \ell)$ for a particular ℓ , we first assume a “baseline” behavior for the sensors, i.e., we make an assumption about the set of sensors that are awake at time k given that $b_{k-1} = b$. We consider two possibilities:

- 1) That all sensors are asleep.
- 2) That the set of sensors awake is selected through a greedy algorithm. In other words, the sensor that causes the largest decrease in expected tracking cost is added to the awake set until any further reduction due to a single sensor is less than c . The expected tracking cost can be evaluated through the use of Monte Carlo simulation (repeatedly simulating our system from time $k-1$ to time k) to avoid the need for numerical integration.

Starting with this set of awake sensors, the value of $T^\Delta(b, \ell)$ is then computed as the absolute difference in expected tracking cost incurred by changing the state of sensor ℓ . Again, Monte Carlo simulation can be used to evaluate the change in expected tracking cost. We can think of this procedure as linearizing the tracking cost about some baseline behavior.

If \mathcal{B} is not finite, then a parameterized version of T^Δ can be computed instead. We choose \tilde{m} elements of $\mathcal{B} - \mathcal{T}$ and evaluate T^Δ at these points. The value of T^Δ at all other values of $b \in \mathcal{B} - \mathcal{T}$ can be computed via an interpolation algorithm. Recall that only an FCR policy is appropriate in the infinite state case, since solving the Q_{MDP} Bellman equation for an infinite number of point mass distributions is infeasible.

C. Learning approach

In this section, we describe an alternative learning-based approach. For ease of exposition, suppose that \mathcal{B} is a finite space. Then our probability measure p_k can be characterized by a probability mass function. We refer to this probability mass function as \mathbf{p}_k (a row vector). Define $\hat{a}_{k,\ell}$ to be the approximated expected increase in tracking cost due to sensor ℓ sleeping at time k as

$$\hat{a}_{k,\ell} = \sum_{b \neq \mathcal{T}} \mathbf{p}_{k-1}(b) T^\Delta(b, \ell) \quad (30)$$

Ideally, we would like this approximation to be equal to the actual expected increase in tracking cost due to sensor ℓ sleeping. Unfortunately, we do not have access to actual tracking costs at time k since b_k is not known exactly. However, we do have access to \mathbf{p}_k , \mathbf{r}_k , and \mathbf{p}_{k-1} . It is therefore possible to estimate the tracking cost as

$$\int_{\mathcal{B}} d(b, \beta^*(\mathbf{p}_k)) \mathbf{p}_k(db) \quad (31)$$

For example, if Hamming cost is being used, then we can estimate the tracking cost as

$$1 - \max_b p_k(\{b\}) \quad (32)$$

and if squared Euclidean distance is being used we can estimate the tracking cost using the variance of the measure p_k . Next we describe how we learn T^Δ by solving a least squares problem.

Determining an estimate of the *increase* in the tracking cost due to the sleeping of sensor ℓ at time k , denoted $a_{k,\ell}$, depends on the value of $r_{k,\ell}$. If $r_{k,\ell} = 0$, we ignore the observation from sensor ℓ and generate a new version of \mathbf{p}_k called \mathbf{p}'_k . We can compute $a_{k,\ell}$ as

$$a_{k,\ell} = \sum_{b \neq \mathcal{T}} \mathbf{p}'_k(b) d(b, \beta^*(\mathbf{p}'_k)) - \sum_{b \neq \mathcal{T}} \mathbf{p}_k(b) d(b, \beta^*(\mathbf{p}_k)) \quad (33)$$

If on the other hand $r_{k,\ell} > 0$, we first generate an object location b'_k according to \mathbf{p}_k and then generate an observation according to the probability measure $\sigma_{b'_k}$. This observation is used to generate a new distribution \mathbf{p}'_k from \mathbf{p}_k . Then we compute $a_{k,\ell}$ as

$$a_{k,\ell} = \sum_{b \neq \mathcal{T}} \mathbf{p}_k(b) d(b, \beta^*(\mathbf{p}_k)) - \sum_{b \neq \mathcal{T}} \mathbf{p}'_k(b) d(b, \beta^*(\mathbf{p}'_k)) \quad (34)$$

We now have an approximation sequence $\hat{a}_{k,\ell}$ and an observation sequence $a_{k,\ell}$. At time $k - 1$, our goal is to choose T^Δ to minimize

$$\mathbb{E} [(\hat{a}_{k,\ell} - a_{k,\ell})^2] \quad (35)$$

We apply the Robbins-Monro algorithm, a form of stochastic gradient descent, to this problem in order to recursively compute a sequence of T^Δ that will hopefully solve this minimization problem for large k . The update equation is

$$T_k^\Delta(b, \ell) = T_{k-1}^\Delta(b, \ell) - 2\alpha_k \mathbb{1}\{b \neq \mathcal{T}\} \mathbf{p}_{k-1}(b) (\hat{a}_{k,\ell} - a_{k,\ell}) \quad (36)$$

where α_k is a step size. Note that $\mathbb{1}\{b \neq \mathcal{T}\} \mathbf{p}_{k-1}(b)$ is the gradient of $\hat{a}_{k,\ell}$ with respect to $T^\Delta(b, \ell)$.

Using a constant step size in our simulations, we could only observe small oscillations in the values of T^Δ . It is unclear whether there are conditions under which the local or global convergence of this learning algorithm is guaranteed. The difficulty is that the observations we are trying to model depend on the model itself. The problem is reminiscent of optimistic policy iteration (see [18]), the convergence properties of which are little understood. We have left a proof of convergence for future work. It should be pointed out that the algorithm will likely converge more slowly for a two-dimensional network than a one-dimensional network. The reason is that in two dimensions it is easier for an object to avoid visiting an object location state and causing an update to that particular value of T^Δ .

If \mathcal{B} is not finite, then we can again parameterize T^Δ as in the previous section. The Robbins-Monro algorithm can be applied in this context as well, although the gradient expressions will depend on the type of interpolation used.

D. A Lower Bound

Unfortunately, deriving a lower bound is generally difficult for the considered problem. However, in this section we derive a lower bound for the special case of a discrete state space with Gaussian observations. Our approach is similar to [13] in which we considered a related scheduling problem. The idea is to combine the observable-after-control assumption with a separable lower bound on the tracking cost as we demonstrate in what follows.

Given the current belief \mathbf{p}_k , an action vector \mathbf{u}_k , and the current residual sleep times vector \mathbf{r}_k , the expected tracking cost can be written as:

$$\begin{aligned} E[d(\hat{b}_{k+1}, b_{k+1}) | \mathbf{p}_k, \mathbf{u}_k, \mathbf{r}_k] &= \sum_{j=1}^m \Pr[\hat{b}_{k+1} \neq j | \mathbf{p}_k, \mathbf{u}_k, \mathbf{r}_k, b_{k+1} = j] \Pr[b_{k+1} = j | \mathbf{p}_k, \mathbf{u}_k] \\ &= \sum_{i=1}^m \mathbf{p}_k(i) \sum_{j=1}^m p(b_{k+1} = j | b_k = i) \Pr[\hat{b}_{k+1} \neq j | \mathbf{p}_k, \mathbf{u}_k, \mathbf{r}_k, b_{k+1} = j] \end{aligned} \quad (37)$$

When awake, the sensors observations are Gaussian, i.e.,

$$s_{k,\ell} \sim \mathcal{N}\left(\frac{10}{(\nu_\ell - b_k)^2 + 1}, 1\right) \quad (38)$$

where ν_ℓ is the location of sensor ℓ .

Defining,

$$P(E|H_j) \triangleq \Pr[\hat{b}_{k+1} \neq j | \mathbf{p}_k, \mathbf{u}_k, \mathbf{r}_k, b_{k+1} = j]$$

which is a conditional error probability for a multiple hypothesis testing problem with m hypotheses, each corresponding to a different mean vector contaminated with white Gaussian noise. Conditioned on H_j , the observation model is:

$$H_j : \mathbf{s}(\ell) = (m_j(\ell) + w)\mathbb{1}\{r_{k+1,\ell} = 0\} + \varepsilon\mathbb{1}\{r_{k+1,\ell} > 0\} \quad (39)$$

where $\mathbf{s}(\ell)$ is the ℓ -th entry of an $n \times 1$ vector \mathbf{s} denoting the received signal strength at the n sensors, m_j is the mean received signal strength when the target is at state j (j -th hypothesis) and w is a zero mean white Gaussian Noise, i.e. $w \sim \mathcal{N}(0, \sigma^2)$. According to (39), if awake at the next time step, sensor ℓ gets a Gaussian observation that depends on the future target location, and an erasure, otherwise. Since the current belief is \mathbf{p}_k , the prior for the j -th hypothesis is $\pi_j = [\mathbf{p}_k P]_j$.

The error event E can be written as the union of pairwise error regions as

$$p(E|H_j) = \Pr[\cup_{k \neq j} \zeta_{kj}] \quad (40)$$

where

$$\zeta_{kj} = \{\mathbf{s} : L_{kj}(\mathbf{s}) > \frac{\pi_j}{\pi_k}\}$$

is the region of observations for which the k -th hypothesis H_k is more likely than the j -th hypothesis H_j , and where

$$L_{kj} \triangleq \frac{f(\mathbf{s}|H_k)}{f(\mathbf{s}|H_j)}$$

denotes the likelihood ratio for H_k and H_j .

Using standard analysis for likelihood ratio tests [22], [23], it is not hard to show that:

$$p(\zeta_{kj}|H_j) = Q\left(\frac{\mathbf{d}_{kj}}{2} + \frac{\ln \frac{\pi_j}{\pi_k}}{\mathbf{d}_{kj}}\right) \quad (41)$$

where $\mathbf{d}_{kj}^2 = \frac{\Delta \mathbf{m}_{kj}^T \Delta \mathbf{m}_{kj}}{\sigma^2}$, $\Delta \mathbf{m}_{kj} = \mathbf{m}_k - \mathbf{m}_j$, and $Q(\cdot)$ is the normal distribution Q -function. The quantity \mathbf{d}_{kj} plays the role of distance between the two hypothesis and hence depends on the difference of their corresponding mean vectors and the noise variance σ^2 . Hence, \mathbf{d}_{kj} is a function of the next step residual sleep vector \mathbf{r}_{k+1} . To highlight this dependence, we will sometimes use the notation $\mathbf{d}_{kj}(\mathbf{r})$ when needed. Note that, for different values of k and j , ζ_{kj} are not generally disjoint but allow us to lower bound the error probability in terms of pairwise error probabilities, namely, a lower bound can be written as:

$$p(E|H_j) \geq \max_{k \neq j} p(\zeta_{kj}|H_j) \quad (42)$$

And we can readily lower bound the expected tracking error:

$$\begin{aligned} E[d(\hat{b}_{k+1}, b_{k+1})|\mathbf{p}_k, \mathbf{u}_k] &\geq \sum_{i=1}^m \mathbf{p}_k(i) \sum_{j=1}^m p(b_{k+1} = j|b_k = i) \max_{k \neq j} p(\zeta_{kj}|H_j) \\ &= \sum_{i=1}^m \mathbf{p}_k(i) \sum_{j=1}^m p(b_{k+1} = j|b_k = i) \max_{k \neq j} Q\left(\frac{\mathbf{d}_{kj}}{2} + \frac{\ln \frac{\pi_j}{\pi_k}}{\mathbf{d}_{kj}}\right) \end{aligned} \quad (43)$$

Next we separate out the effect of each sensor on the tracking error:

$$\begin{aligned} E[d(\hat{b}_{k+1}, b_{k+1})|\mathbf{p}_k, \mathbf{u}_k, \mathbf{r}_k] &\stackrel{(a)}{\geq} \mathbb{1}\{r_{k+1,\ell} = 0\} E[d(\hat{b}_{k+1}, b_{k+1})|\mathbf{p}_k, \mathbf{r}_{k+1} = \mathbf{0}] \\ &\quad + \mathbb{1}\{r_{k+1,\ell} > 0\} E[d(\hat{b}_{k+1}, b_{k+1})|\mathbf{p}_k, r_{k+1,i} = 0 \ \forall i \neq \ell] \text{ for every } \ell \end{aligned} \quad (44)$$

where $\mathbf{0}$ is the all zero vector designating that all sensors will be awake at the next time slot. The inequality in (a) follows from the fact that if we separate out the effect of the ℓ -th sensor we get a better tracking performance when all the remaining sensors are awake. Since this holds for every ℓ , a lower bound on the expected tracking error can be written as a convex combination of all sensors contributions:

$$\begin{aligned} E[d(\hat{b}_{k+1}, b_{k+1})|\mathbf{p}_k, \mathbf{u}_k, \mathbf{r}_k] &\geq \sum_{\ell=1}^n \lambda_\ell(\mathbf{p}_k) \left\{ \mathbb{1}\{r_{k+1,\ell} = 0\} E[d(\hat{b}_{k+1}, b_{k+1})|\mathbf{p}_k, \mathbf{r}_{k+1} = \mathbf{0}] \right. \\ &\quad \left. + \mathbb{1}\{r_{k+1,\ell} > 0\} E[d(\hat{b}_{k+1}, b_{k+1})|\mathbf{p}_k, r_{k+1,i} = 0 \ \forall i \neq \ell] \right\} \end{aligned} \quad (45)$$

where $\sum_{\ell} \lambda_\ell(\mathbf{p}_k) = 1$.

Let $\mathbf{0}_{-\ell}$ denote a vector of length n with all entries equal to zero except for the ℓ -th entry which can be anything greater than 0. Then replacing from (43),

$$E[d(\hat{b}_{k+1}, b_{k+1}) | \mathbf{p}_k, \mathbf{u}_k, \mathbf{r}_k] \geq \sum_{\ell=1}^n \lambda_{\ell}(\mathbf{p}_k) \left\{ \mathbb{I}_{\{r_{k+1,\ell}=0\}} \sum_{i=1}^m \mathbf{p}_k(i) \sum_{j=1}^m p(b_{k+1}=j | b_k=i) \max_{k \neq j} Q \left(\frac{\mathbf{d}_{kj}(\mathbf{0})}{2} + \frac{\ln \frac{\pi_j}{\pi_k}}{\mathbf{d}_{kj}(\mathbf{0})} \right) + \mathbb{I}_{\{r_{k+1,\ell} > 0\}} \sum_{i=1}^m \mathbf{p}_k(i) \sum_{j=1}^m p(b_{k+1}=j | b_k=i) \max_{k \neq j} Q \left(\frac{\mathbf{d}_{kj}(\mathbf{0}_{-\ell})}{2} + \frac{\ln \frac{\pi_j}{\pi_k}}{\mathbf{d}_{kj}(\mathbf{0}_{-\ell})} \right) \right\} \quad (46)$$

To simplify notation, we introduce the following 2 quantities:

$$T_0(\mathbf{p}; i, \ell) \triangleq \sum_{j=1}^m p(b_{k+1}=j | b_k=i) \max_{k \neq j} Q \left(\frac{\mathbf{d}_{kj}(\mathbf{0})}{2} + \frac{\ln \frac{[\mathbf{p}P]_j}{[\mathbf{p}P]_k}}{\mathbf{d}_{kj}(\mathbf{0})} \right)$$

$$T(\mathbf{p}; i, \ell) \triangleq \sum_{j=1}^m p(b_{k+1}=j | b_k=i) \max_{k \neq j} Q \left(\frac{\mathbf{d}_{kj}(\mathbf{0}_{-\ell})}{2} + \frac{\ln \frac{[\mathbf{p}P]_j}{[\mathbf{p}P]_k}}{\mathbf{d}_{kj}(\mathbf{0}_{-\ell})} \right)$$

Intuitively, $T_0(\mathbf{p}; i, \ell)$ represents the contribution of sensor ℓ to the total expected tracking cost when the underlying state is i , the belief is \mathbf{p} and when all sensors are awake. On the other hand $T(\mathbf{p}; i, \ell)$ is the ℓ -th sensor contribution when it is asleep and all the other sensors are awake.

Now if we assume that the target will be perfectly observable after taking the sleeping action, a lower bound on the total cost can be obtained from the solution of the following Bellman equation:

$$J(\mathbf{p}, \mathbf{r}_0) = \sum_{\ell} J^{\ell}(\mathbf{p}, r_{0,\ell}) \quad (47)$$

where,

$$J^{\ell}(\mathbf{p}, r_{0,\ell}) = \min_{u_{\ell}} \left\{ \mathbb{I}_{\{r_{1,\ell}=0\}} \left(\sum_b \mathbf{p}(b) \lambda_{\ell} T_0(\mathbf{p}; b, \ell) + c \sum_{i=1}^m [\mathbf{p}P]_i + \sum_{i=1}^m [\mathbf{p}P]_i J(\mathbf{e}_i, 0) \right) + \mathbb{I}_{\{r_{1,\ell} > 0\}} \left(\sum_b \mathbf{p}(b) \lambda_{\ell} T(\mathbf{p}; b, \ell) + \sum_{i=1}^m [\mathbf{p}P]_i J(\mathbf{e}_i, u_{\ell}) \right) \right\} \quad (48)$$

Note that if we can solve the equation above for $\mathbf{p} = \mathbf{e}_i$ for all $i \in \{1, \dots, m\}$, then it is straightforward to find the solution for all other values of \mathbf{p} . We therefore focus on specifying the value function at those points. Since this is the case, we further simplify our notation and use $T(i, \ell)$ and $\lambda(i, \ell)$ as shorthand for $T(\mathbf{e}_i; i, \ell)$ and $\lambda_{\ell}(\mathbf{e}_i)$, respectively. Also since an action only needs to be made when the sensor wakes up, we only need to define actions at $r_{0,\ell} = 0$. Observing that

$$J^{\ell}(\mathbf{e}_j, u) = \lambda(j, \ell) T(j, \ell) + \sum_{i=1}^m [e_j P]_i J^{\ell}(\mathbf{e}_i, u-1) \quad \forall u > 1 \quad (49)$$

and

$$J^\ell(e_j, 1) = \lambda(j, \ell)T_0(j, \ell) + c \sum_{i=1}^m [e_j P]_i + \sum_{i=1}^m [e_j P]_i J^\ell(e_i, 0) \quad (50)$$

we recursively substitute from (49) and (50) in (48) until the system reaches $(e_i, 0)$. We can see that a lower bound on the value function of sensor ℓ can be obtained as a solution of the following minimization problem over u_ℓ^b , where u_ℓ^b is the control action for sensor ℓ given a belief state e_b

$$J^\ell(e_b) = \min_u \left\{ \sum_{j=0}^{u-1} \sum_{i=1}^m [e_b P^j]_i \lambda(i, \ell) T(i, \ell) + \sum_{i=1}^m [e_b P^u]_i \lambda(i, \ell) T_0(i, \ell) \right. \\ \left. + c \sum_{i=1}^m [e_b P^{u+1}]_i + \sum_{i=1}^m [e_b P^{u+1}]_i J^\ell(e_i) \right\} \quad (51)$$

Equation (51) together with (47) define a lower bound on the total expected cost. To further tighten the bound we can now optimize over a matrix Λ for every value of c , where $\Lambda(c)$ is an $m \times n$ matrix with the (i, ℓ) entry equal to $\lambda(i, \ell)$, i.e., $\Lambda(c) = \{\lambda(i, \ell)\}$. Hence,

$$J(e_b) = \max_{\Lambda(c)} \sum_{\ell=1}^n \min_{u_\ell^b} \left\{ \sum_{j=0}^{u-1} \sum_{i=1}^m [e_b P^j]_i \lambda(i, \ell) T(i, \ell) + \sum_{i=1}^m [e_b P^u]_i \lambda(i, \ell) T_0(i, \ell) \right. \\ \left. + c \sum_{i=1}^m [e_b P^{u+1}]_i + \sum_{i=1}^m [e_b P^{u+1}]_i J^\ell(e_i) \right\} \quad (52)$$

subject to $\Lambda \mathbf{1}_n = \mathbf{1}_m$

where $\mathbf{1}_m$ is a column vector of all ones of length m . A closed form solution for (52) cannot be obtained, and hence, we solve for $J(e_b)$ numerically. First, we fix Λ and use policy iteration [18] to solve for the control of each sensor at each state. Then, we change Λ and repeat the process. The envelope of the generated value functions (corresponding to different instants of Λ) is hence a lower bound on the optimal value function.

IV. NUMERICAL RESULTS

In this section, we show some simulation results illustrating the performance of the policies we derived in previous sections. These results will be for one-dimensional sensor networks, but the general behavior should extend to two-dimensional networks. In each simulation run, the object was initially placed at the center of the network and the location of the object was made known to each sensor. A simulation run concluded when the object left the network. The results of many simulation runs were then averaged to compute an average tracking cost and an average energy cost. To allow for easier interpretation of our results, we then normalized our costs by dividing by the *expected* time the object spends in the network.

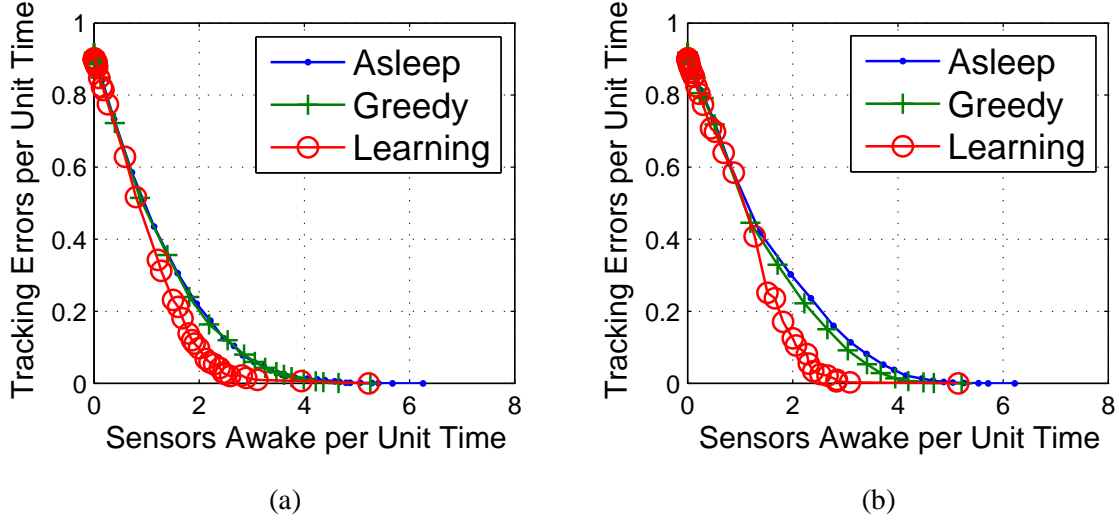


Fig. 1. Tradeoff curves for Network A: (a) Q_{MDP} policies; (b) FCR policies

We refer to these normalized costs as costs per unit time, even though the true costs per unit time would use the *actual* times the object spent in the network (the difference between the two was found to be small).

For the non-learning policies, the value of $T^\Delta(b, \ell)$ for each b and ℓ was generated using 200 Monte Carlo simulations. The results of 50 simulation runs were averaged when plotting the curves. For the learning policies, the values for T^Δ were initialized to those obtained from the non-learning approach using greedy sensor selection as a baseline. A constant step size of 0.01 was used in the learning algorithm. First, 100 simulation runs were performed but the results were not recorded while the values for T^Δ stabilized. Then an additional 50 simulation runs were performed (T^Δ continued to be updated) and these results were averaged when plotting curves. In the case of Q_{MDP} learning policies, computation time was saved by performing policy iteration only after every fifth simulation run.

We first consider a simple network that we term Network A. This is a one-dimensional network with 41 possible object locations where the object moves with equal probability either one to the left or one to the right in each time step. There is a sensor at each of the 41 object locations that makes (when awake) a binary observation that determines without error whether the object is at that location. Hamming cost is used for the tracking cost.

For Network A, we illustrate the performance of the Q_{MDP} versions of our policies in Figure 1(a) and the FCR versions of our policies in Figure 1(b).

The curves labeled “Asleep” are for the nonlearning approach for computing T^Δ where we assume that

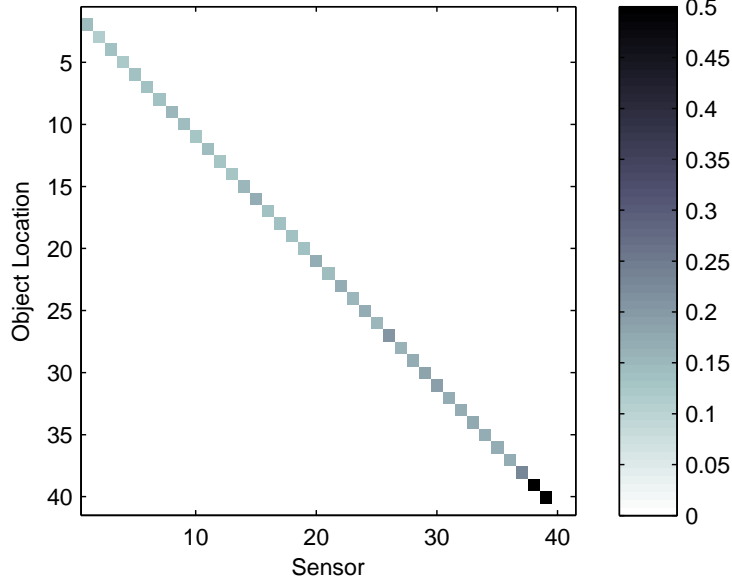


Fig. 2. The final matrix for T^Δ for the Q_{MDP} learning policy and small c for Network A.

all sensors are asleep as a baseline. The curves labeled “Greedy” are for the nonlearning approach for computing T^Δ where we use a greedy algorithm to determine our baseline. The curves labeled “Learning” employ our learning algorithm for computing T^Δ .

From the tradeoff curves, it is apparent that using the learning algorithm to compute T^Δ results in improved performance. A close inspection of Figures 1(a) and 1(b) will reveal that the Q_{MDP} policies perform somewhat better than their FCR counterparts. This is consistent with what was observed in [1].

It is instructive to consider the final matrix of values for $T^\Delta(b, \ell)$ that was obtained at the end of all learning algorithm simulations. In Figures 2 and 3 we plot this matrix for the Q_{MDP} learning policy simulations for the smallest c and for the largest c used in simulation, respectively. In Figure 2, it is evident that only a single sensor has an impact for each value of b . Due to the way our simulations worked, it is the sensor to the left that has the impact, but it could just as easily be the sensor to the right of the current object position. The fact that most of the nonzero values of the matrix are less than 0.5 reflects the fact that the sensor to the right of the current object location might wake up due to a sleep time selected at a previous time step. In Figure 3, it is evident that the sensors on either side of the current object location (which is actually not known since Figure 3 corresponds to the case where no sensors are awake) appear to have a major impact on the tracking cost. There are nonzero values off the

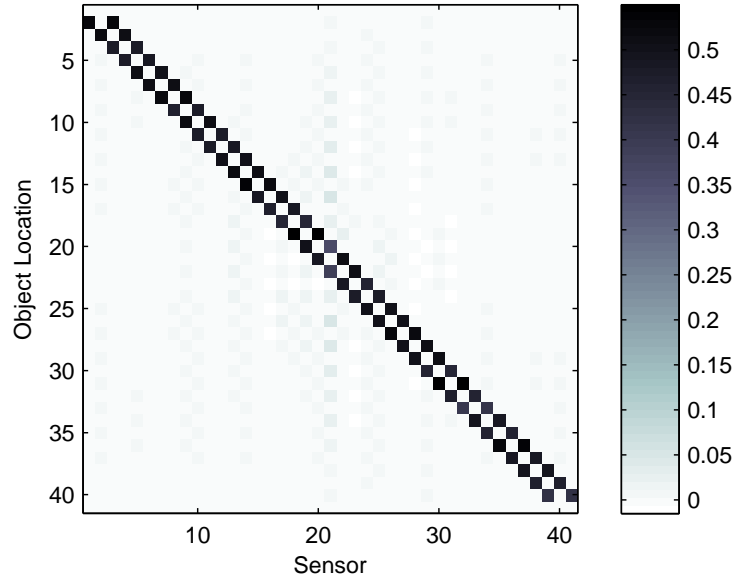


Fig. 3. The final matrix for T^Δ for the Q_{MDP} learning policy and large c for Network A.

TABLE I
OBJECT MOVEMENT FOR NETWORK B.

<i>Change in Position</i>	0	1	2	3
<i>Probability</i>	0.3125	0.2344	0.0938	0.0156

two main diagonals due to probabilistic nature of the learning process when the actual object location is not known.

We now consider a new one-dimensional network termed Network B. The possible object locations are located on the integers from 1 to 21. The object moves according to a random walk anywhere from three steps to the left to three steps to the right in each time step. The distribution of these movements is given in table I. The change in position indicate movement by a corresponding number of steps to the right or to the left. There are 10 sensors in this network so that $m \neq n$. The locations of the sensors are given in Table II and awake sensors make Gaussian observations as in (38).

Results for the Q_{MDP} and FCR versions of our policies are shown in Figures 4(a) and 4(b), respectively. The results confirm the same general trends observed for Network A. The figures also show our derived lower bound on the energy-tracking tradeoff using the approach described in Sec. III-D. Not surprisingly,

TABLE II
SENSOR LOCATIONS FOR NETWORK B.

Sensor	1	2	3	4	5	6	7	8	9	10
Location	1.36	1.61	3.91	8.09	11.96	13.39	13.52	13.66	16.60	18.68

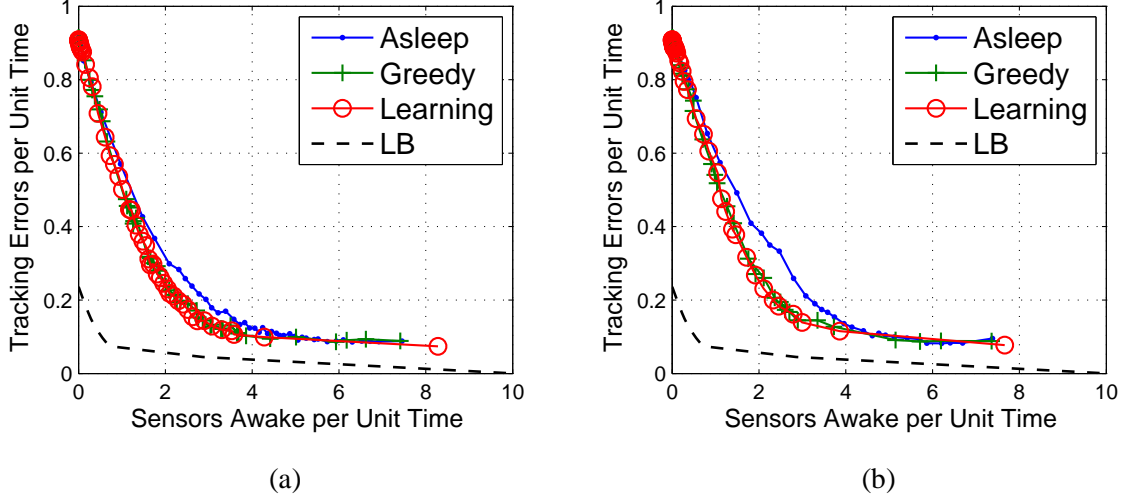


Fig. 4. Tradeoff curves for Network B and a lower bound: (a) Q_{MDP} policies; (b) FCR policies

the lower bound is particularly loose at the high tracking cost regime, yet the gap is reasonably small for the low tracking error region. This is expected since the lower bound uses an all-awake assumption to lower bound the contribution of each sensor to the tracking error. However, it is worth mentioning that we can exactly compute the saturation point for the optimal scheduling policy, which matches the saturation limit of the shown curves, since every policy has to eventually meet the all-asleep performance curve when the energy cost per sensor is high. At that point, all sensors are put to sleep and hence the target estimate can only be based on prior information. The small gap at the low tracking error regime combined with the aforementioned saturation effect highlight good performance for our sleeping policies. For illustration, we plot the matrix for T^Δ for the Q_{MDP} learning policy simulations for the smallest c and for the largest c when the object moves according to a symmetric random walk in Figures 5 and 6, respectively. Note the difference between the rows corresponding to object locations 7 and 8 in Figure 5. Examining the sensor locations, we see that sensor 4 is located at 8.09. This sensor is useful for distinguishing between object locations 6 and 8 (for an initial object position of 7) but is of less value for distinguishing between object locations 7 and 9 (for an initial object position of 8). This is evidenced

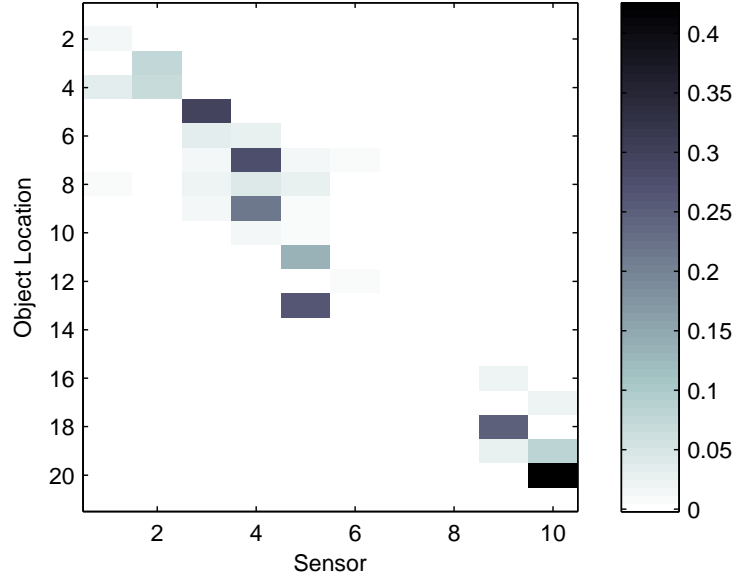


Fig. 5. The final matrix for T^Δ for the Q_{MDP} learning policy and small c for Network B.

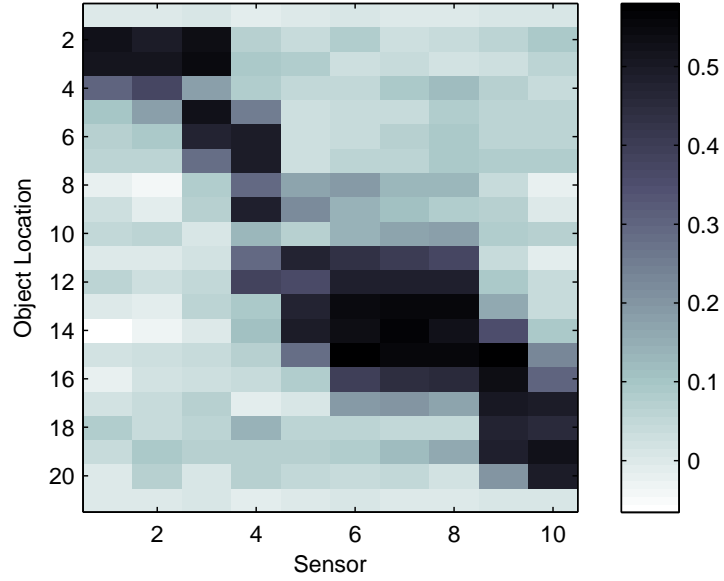


Fig. 6. The final matrix for T^Δ for the Q_{MDP} learning policy and large c for Network B.

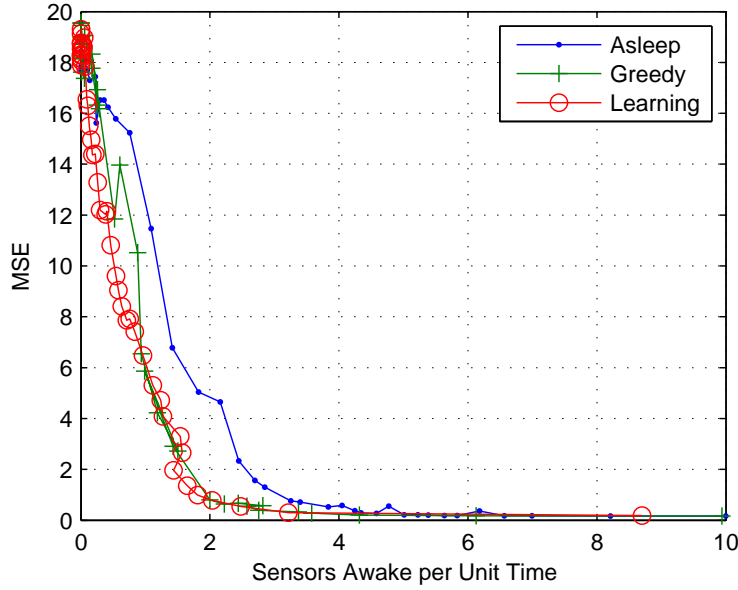


Fig. 7. Tradeoff curves for FCR policies for Network C.

in the figure as a large value for $T^\Delta(7, 4)$ and a small value for $T^\Delta(8, 4)$.

To demonstrate that our techniques can be applied to an object that moves on a continuum, we define a new network, Network C. This network is identical to Network B except for two changes. First, the object can take locations anywhere on the interval $[1, 21]$. Second, the object moves according to Brownian motion with the change in position between time steps having a Gaussian distribution with mean zero and variance 1. As mentioned earlier, only FCR policies can be generated for this type of network. Values of T^Δ were computed for each integer-valued object location on $[1, 21]$ and linear interpolation used to compute values of T^Δ for other object locations. Since continuous distributions cannot be easily stored, particle filtering techniques were employed (e.g., see [19]). The number of particles used was 512 and resampling was performed at each time step. As is consistent with particle filtering, in generating the sleep times the computation of future probability distributions was approximated through Monte Carlo movement of the particles. The number of simulation runs that were averaged for each data point was increased to 200 for these simulations.

Tradeoff curves for Network C are shown in Figure 7. Although the tradeoff curves are less smooth than before, this figure illustrates performance trends similar to those already seen. The reason the curves are not as smooth is that occasionally the particle filter would fail to keep track of the distribution with

sufficient accuracy. This would cause the network to lose track of the object and cause abnormally bad tracking for that simulation run. These outliers were not removed when generating the tradeoff curves. A recovery mechanism would need to be added to the sleeping policies to overcome this limitation of particle filters.

V. CONCLUSION

In this paper, we considered energy-efficient tracking of an object moving through a network of wireless sensors. While an optimal solution could not be found, it was possible to design suboptimal, yet efficient, sleeping solutions for general motion, sensing, and cost models. We proposed Q_{MDP} and FCR approximate policies, where in the former, the system is assumed to be perfectly observable after control, and in the latter, to be totally unobservable. We combined these approximations with a decomposition of the optimization problem into simpler per-sensor subproblems, and developed learning and non-learning based approaches to compute the parameters of each subproblem. The learning-based Q_{MDP} policies were shown to provide the best energy-tracking tradeoff. In the low tracking error regime, our sleeping policies approach a derived lower bound on the optimal energy-tracking tradeoff.

Avenues for future research include developing distributed sleeping strategies in the absence of central control and solving the tracking problem for unknown or partially known object movement statistics.

REFERENCES

- [1] J. A. Fuemmeler and V. V. Veeravalli, "Smart sleeping policies for energy efficient tracking in sensor networks," *IEEE Transactions on Signal Processing*, vol. 56, no. 5, pp. 2091–2101, May 2008.
- [2] R. R. Brooks, P. Ramanathan, and A. M. Sayeed, "Distributed target classification and tracking in sensor networks," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1163–1171, Aug. 2004.
- [3] S. Balasubramanian, I. Elangovan, S. K. Jayaweera, and K. R. Namuduri, "Distributed and collaborative tracking for energy-constrained ad-hoc wireless sensor networks," in *IEEE Wireless Communications and Networking Conference*, vol. 3, Mar. 2004, pp. 1732–1737.
- [4] R. Gupta and S. R. Das, "Tracking moving targets in a smart sensor network," in *IEEE 58th Vehicular Technology Conference*, vol. 5, Oct. 2003, pp. 3035–3039.
- [5] H. Yang and B. Sikdar, "A protocol for tracking mobile targets using sensor networks," in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003, pp. 71–81.
- [6] Y. Xu, J. Winter, and W.-C. Lee, "Prediction-based strategies for energy saving in object tracking sensor networks," in *Proceedings of the 2004 IEEE International Conference on Mobile Data Management*, 2004, pp. 346–357.
- [7] L. Yang, C. Feng, J. W. Rozenblit, and J. Peng, "A multi-modality framework for energy efficient tracking in large scale wireless sensor networks," in *Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control*, Apr. 2006, pp. 916–921.

- [8] C. Gui and P. Mohapatra, "Power conservation and quality of surveillance in target tracking sensor networks," in *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MOBICOM)*, Sep. 2004, pp. 129–143.
- [9] —, "Virtual patrol: A new power conservation design for surveillance using sensor networks," in *Fourth International Symposium on Information Processing in Sensor Networks (IPSN)*, Apr. 2005, pp. 246–253.
- [10] N. A. Vasanthi and S. Annadurai, "Energy saving schedule for target tracking sensor networks to maximize the network lifetime," in *First International Conference on Communication System Software and Middleware*, Jan. 2006, pp. 1–8.
- [11] D. A. Castanon, "Approximate dynamic programming for sensor management," in *36th conference on decision and control (CDC)*, 1997, pp. 1202–1207.
- [12] G. K. Atia, J. A. Fuemmeler, and V. V. Veeravalli, "Sensor scheduling for energy-efficient target tracking in sensor networks," *Accepted to Asilomar Conference on Signals, Systems, and Computers*, July 2010.
- [13] G. K. Atia, V. V. Veeravalli, and J. A. Fuemmeler, "Sensor scheduling for energy-efficient target tracking in sensor networks," *Submitted to IEEE Transactions on Signal Processing*, July 2010.
- [14] D. Aberdeen, "A (revised) survey of approximate methods for solving partially observable Markov decision processes," National ICT Australia, Canberra, Australia, Tech. Rep., Dec. 2003. [Online]. Available: <http://users.rsise.anu.edu.au/~daa/papers.html>.
- [15] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *Proceedings of the Twelfth International Conference on Machine Learning*, 1995, pp. 362–370.
- [16] G. Monahan, "A survey of partially observable markov decision processes: theory, models, and algorithms," *Management Science*, vol. 28, pp. 1–16, 1982.
- [17] M. Hauskrecht, "Value-function approximations for partially observable markov decision processes," *Journal of Artificial Intelligence Research (JAIR)*, vol. 13, pp. 33–94, 2000.
- [18] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Belmont, MA: Athena Scientific, 2007.
- [19] A. Doucet, N. de Freitas, and N. Gordon, Eds., *Sequential Monte Carlo Methods in Practice*. New York, NY: Springer-Verlag, 2001.
- [20] A. Cassandra, M. L. Littman, and N. L. Zhang, "Incremental pruning: A simple, fast, exact algorithm for partially observable markov decision processes," in *Thirteenth Annual Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1997, pp. 54–61.
- [21] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: scaling up," in *Twelfth International Conference on Machine Learning*, 1995, pp. 362–370.
- [22] H. V. Poor, *An Introduction to Signal Detection and Estimation (2nd ed.)*. New York, NY, USA: Springer-Verlag New York, Inc., 1994.
- [23] B. C. Levy, *Principles of Signal Detection and Parameter Estimation*. Springer Publishing Company, Incorporated, 2008.